```
#!/usr/bin/env python3
# USING PYTHON 3.6.7

# Homework Number: 6
# Name: Michael Cupka
# ECN Login: mcupka
# Due Date: March 7, 2019

# This program takes in a string in a file and outputs the SHA-512 hash of that
string in hex format to an output file

import os
import sys
from BitVector import *

# constants for sha-512
K = ['428a2f98d728ae22', '7137449123ef65cd', 'b5c0fbcfec4d3b2f', 'e9b5dba58189dbbc',
'3956c25bf348b538', '59f111f1b605d019', '923f82a4af194f9b', 'ab1c5ed5da6d8118',
'd807aa98a3030242', '12835b0145706fbe', '243185be4ee4b28c', '550c7dc3d5ffb4e2',
'72be5d74f27b896f', '80deb1fe3b1696b1', '9bdc06a725c71235', 'c19bf174cf692694',
'e49b69c19ef14ad2', 'efbe4786384f25e3', '0fc19dc68b8cd5b5', '240ca1cc77ac9c65',
'2de92c6f592b0275', '4a7484aa6ea6e483', '5cb0a9dcbd41fbd4', '76f988da831153b5',
'983e5152ee66dfab', 'a831c66d2db43210', 'b00327c898fb213f', 'bf597fc7beef0ee4',
'c6e00bf33da88fc2', 'd5a79147930aa725', '06ca6351e003826f', '142929670a0e6e70',
'27b70a8546d22ffc', '2e1b21385c26c926', '4d2c6dfc5ac42aed', '53380d139d95b3df',
'650a73548baf63de', '766a0abb3c77b2a8', '81c2c92e47edaee6', '92722c851482353b',
'a2bfe8a14cf10364', 'a81a664bbc423001', 'c24b8b70d0f89791', 'c76c51a30654be30',
'd192e819d6ef5218', 'd69906245565a910', 'f40e35855771202a', '106aa07032bbd1b8',
'19a4c116b8d2d0c8', '1e376c085141ab53', '2748774cdf8eeb99', '34b0bcb5e19b48a8',
'391c0cb3c5c95a63', '4ed8aa4ae3418acb', '5b9cca4f7763e373', '682e6ff3d6b2b8a3',
'748f82ee5defb2fc', '78a5636f43172f60', '84c87814a1f0ab72', '8cc702081a6439ec',
'90befffa23631e28', 'a4506cebde82bde9', 'bef9a3f7b2c67915', 'c67178f2e372532b',
'ca273eceea26619c', 'd186b8c721c0c207', 'eada7dd6cde0eb1e', 'f57d4f7fee6ed178',
'06f067aa72176fba', '0a637dc5a2c898a6', '113f9804bef90dae', '1b710b35131c471b',
'28db77f523047d84', '32caab7b40c72493', '3c9ebe0a15c9bebc', '431d67c49c100d4c',
'4cc5d4becb3e42b6', '597f299cfc657e2a', '5fcb6fab3ad6faec', '6c44198c4a475817']

K_bv = [BitVector(hexstring = k_constant) for k_constant in K]



# function to calculate the hash value of a given input string using sha-512
# This code is adapted and modified based on the sha-256 code in the lecture 15 slides
def sha512hash(input_val):

    # initial h values
    h0 = BitVector(hexstring='6a09e667f3bcc908')
    h1 = BitVector(hexstring='bb67ae8584caa73b')
    h2 = BitVector(hexstring='3c6ef372fe94f82b')
    h3 = BitVector(hexstring='a54ff53a5f1d36f1')
    h4 = BitVector(hexstring='510e527fade682d1')
    h5 = BitVector(hexstring='9b05688c2b3e6c1f')
    h6 = BitVector(hexstring='1f83d9abfb41bd6b')
    h7 = BitVector(hexstring='5be0cd19137e2179')


    # step one
    bv = BitVector(textstring=input_val)
    length = bv.length()
    bv1 = bv + BitVector(bitstring="1")
    length1 = bv1.length()
    howmanyzeros = (896 - length1) % 1024
```

```
    zerolist = [0] * howmanyzeros
    bv2 = bv1 + BitVector(bitlist=zerolist)
    bv3 = BitVector(intVal=length, size=128)
    bv4 = bv2 + bv3

    #initialize words to store message schedule
    words = [None] * 80

    for n in range(0, bv4.length(), 1024):
        block = bv4[n:n+1024]


        # message schedule first 16 words
        words[0:16] = [block[i:i + 64] for i in range(0, 1024, 64)]

        # expanding message schedule. This is where it gets sketchy
        for i in range(16, 80):
            i_minus_2_word = words[i - 2]
            i_minus_15_word = words[i - 15]
            #  The sigma1 function is applied to the i_minus_2_word and the sigma0
function is applied to
            #  the i_minus_15_word:
            sigma0 = (i_minus_15_word.deep_copy() >> 1) ^
(i_minus_15_word.deep_copy() >> 8) ^ (i_minus_15_word.deep_copy().shift_right(7))
            sigma1 = (i_minus_2_word.deep_copy() >> 19) ^ (i_minus_2_word.deep_copy()
>> 61) ^ (i_minus_2_word.deep_copy().shift_right(6))

            words[i] = BitVector(intVal=(int(words[i-16]) + int(sigma1) +
int(words[i-7]) + int(sigma0)) & 0xFFFFFFFFFFFFFFFF, size=64)

        a, b, c, d, e, f, g, h = h0, h1, h2, h3, h4, h5, h6, h7

        # step 3, 80 rounds of processing for each block
        for i in range(80):
            ch = (e & f) ^ ((~e) & g)
            maj = (a & b) ^ (a & c) ^ (b & c)
            sum_a = ((a.deep_copy()) >> 28) ^ ((a.deep_copy()) >> 34) ^
((a.deep_copy()) >> 39)
            sum_e = ((e.deep_copy()) >> 14) ^ ((e.deep_copy()) >> 18) ^
((e.deep_copy()) >> 41)
            t1 = BitVector(intVal=(int(h) + int(ch) + int(sum_e) + int(words[i]) +
int(K_bv[i])) & 0xFFFFFFFFFFFFFFFF, size=64)
            t2 = BitVector(intVal=(int(sum_a) + int(maj)) & 0xFFFFFFFFFFFFFFFF,
size=64)
            h = g
            g = f
            f = e
            e = BitVector(intVal=(int(d) + int(t1)) & 0xFFFFFFFFFFFFFFFF, size=64)
            d = c
            c = b
            b = a
            a = BitVector(intVal=(int(t1) + int(t2)) & 0xFFFFFFFFFFFFFFFF, size=64)

        # step 4, mix the contents of abcdefg with the contents of the hash buffer
        h0 = BitVector(intVal=(int(h0) + int(a)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h1 = BitVector(intVal=(int(h1) + int(b)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h2 = BitVector(intVal=(int(h2) + int(c)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h3 = BitVector(intVal=(int(h3) + int(d)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h4 = BitVector(intVal=(int(h4) + int(e)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h5 = BitVector(intVal=(int(h5) + int(f)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h6 = BitVector(intVal=(int(h6) + int(g)) & 0xFFFFFFFFFFFFFFFF, size=64)
        h7 = BitVector(intVal=(int(h7) + int(h)) & 0xFFFFFFFFFFFFFFFF, size=64)
```

```
        #  Concatenate the contents of the hash buffer to create 512 bit hash
        message_hash = h0 + h1 + h2 + h3 + h4 + h5 + h6 + h7

        #  Get the hex representation
        hash_hex_string = message_hash.getHexStringFromBitVector()
        return hash_hex_string


if __name__ == '__main__':
    input_file = open(sys.argv[1], 'r')
    output_file = open(sys.argv[2], 'w')
    input_val = input_file.read()
    input_file.close()
    sha_512_hash_val = sha512hash(input_val)
    output_file.write(sha_512_hash_val)
```